Universidad Nacional Autónoma de México

Facultad de Ingeniería

Ingeniería en Computación

Ingeniería de Software

Grupo 04

Ing. Orlando Zaldívar Zamorategui

TUTORIAL IS 2025-2 1. Introducción a la Ingeniería de Software. Definición de Ingeniería de Software. (ESCALETA DEL VIDEO)

EQUIPO 2

Benítez Guerrero Alexis
Cabrera Marcos Sergio Ivan
Moreno Muñiz Luis Gerardo

Ciudad Universitaria, Ciudad de México, a 27 / 05 / 2025

1. Ética en la Ingeniería de Software

La ética en la ingeniería de software establece las bases morales y normativas que deben regir la práctica profesional, asegurando el bienestar de los usuarios y de la sociedad. Este compromiso ético es esencial en un campo donde las decisiones pueden afectar directamente la seguridad y privacidad de las personas.

Una de las principales responsabilidades éticas es la creación de software seguro y confiable. Los ingenieros deben evitar la introducción de errores que puedan resultar en fallos catastróficos, especialmente en aplicaciones críticas como las de salud o transporte. Además, los ingenieros de software están llamados a proteger la privacidad de los usuarios. Esto incluye la gestión responsable de datos sensibles y la implementación de prácticas que resguarden la información contra accesos no autorizados.

La ética también implica transparencia en la comunicación de las capacidades del software. Es importante que los ingenieros informen a los clientes y usuarios de cualquier limitación o posible riesgo del sistema para evitar expectativas irreales. En cuanto a la sostenibilidad, los ingenieros deben considerar el impacto ambiental del software, especialmente en lo relacionado con el consumo de energía y el uso de recursos, promoviendo el desarrollo de soluciones más ecológicas.

Otro principio ético es la justicia, lo que significa diseñar software inclusivo y accesible, evitando sesgos que puedan perjudicar a ciertos grupos de personas. La ética en el software también promueve el respeto por la propiedad intelectual, obligando a los ingenieros a evitar el uso de código y recursos no autorizados, además de dar crédito a las fuentes cuando es debido.

El código de ética incluye la obligación de actualizar conocimientos. Los ingenieros deben mantenerse informados de los avances tecnológicos y éticos para no quedarse rezagados en prácticas obsoletas o no éticas. La ética exige que los ingenieros actúen en defensa del interés público incluso por encima de los beneficios personales o empresariales, denunciando cualquier práctica que comprometa la seguridad de los usuarios.

Finalmente, los ingenieros deben participar en la creación y actualización de los estándares éticos, asegurando que estos reflejen las necesidades y desafíos actuales de la industria. (Sommerville, 2011, p. 14)

2. Métodos Ágiles

Los métodos ágiles en la ingeniería de software son enfoques adaptativos que permiten responder rápidamente a los cambios en los requerimientos, promoviendo una mayor flexibilidad en el desarrollo.Estos métodos se basan en la entrega incremental de funciones, lo que facilita la retroalimentación constante de los clientes y permite ajustes continuos según sus necesidades.

Uno de los principios fundamentales del desarrollo ágil es la colaboración constante con el cliente, asegurando que el software evolucione de acuerdo con sus expectativas y condiciones reales.Entre los métodos ágiles más utilizados se encuentran Scrum y Extreme Programming (XP), cada uno con enfoques y estructuras propias que facilitan la planificación y ejecución del desarrollo en equipo.

El desarrollo ágil prioriza a las personas y la comunicación sobre los procesos y las herramientas, promoviendo un ambiente colaborativo que optimiza el trabajo en equipo y la resolución de problemas. La iteración es clave en estos métodos: cada ciclo de desarrollo permite revisar y mejorar el producto, y al final de cada iteración, se tiene una versión funcional del software que se puede evaluar.

En lugar de planificar todos los requisitos desde el inicio, los métodos ágiles permiten definirlos y ajustarlos en el transcurso del proyecto, lo cual es útil en proyectos complejos y con alta incertidumbre. La gestión de riesgos también es un componente esencial, ya que el proceso iterativo permite identificar y abordar problemas en etapas tempranas, reduciendo el costo de los errores.

El enfoque ágil ayuda a fomentar la innovación y la creatividad, ya que los equipos tienen la libertad de probar soluciones y adaptarse sin estar atados a un plan inflexible. Por último, el éxito del método ágil depende en gran medida de la disciplina y la comunicación eficaz dentro del equipo, quienes deben ser capaces de adaptarse y coordinarse en un entorno dinámico y cambiante. (Sommerville, 2011, p. 58)

3. Requerimientos Funcionales y No Funcionales

Los requerimientos funcionales son las funciones específicas que el software debe realizar, como la entrada, el procesamiento y la salida de datos para cumplir con las necesidades del usuario. Por otro lado, los requerimientos no funcionales describen las cualidades o atributos del sistema, tales como la seguridad, rendimiento, eficiencia y usabilidad, aspectos que también son cruciales para la satisfacción del cliente.

La precisión en la definición de estos requerimientos es esencial, ya que un mal entendimiento puede conducir a desarrollos costosos e ineficientes que no cumplen con las expectativas.

La documentación de los requerimientos funcionales se realiza a través de especificaciones detalladas que guían el desarrollo, asegurando que todas las partes involucradas compartan la misma comprensión del sistema.

Los requerimientos no funcionales, aunque a menudo se subestiman, son esenciales para el éxito a largo plazo del software, ya que afectan directamente la experiencia del usuario y la adaptabilidad del sistema.

La ingeniería de requerimientos implica actividades de adquisición y análisis, validación y, en algunos casos, una administración que se adapte a los cambios a medida que el proyecto avanza. La adquisición de requerimientos incluye el

trabajo conjunto con los usuarios y stakeholders para capturar sus necesidades y traducirlas en especificaciones técnicas.
La validación es crucial,ya que permite detectar y corregir posibles erroresen la fase de diseño antes de que se conviertan en problemas costosos. La administración de requerimientos permite hacer ajustes en respuesta a cambios en las necesidades del negocio o del cliente, lo cual es especialmente importante en proyectos de larga duración. Finalmente, tanto los requerimientos funcionales como los no funcionales son necesarios para construir un sistema robusto, eficiente y alineado con las expectativas del cliente. (Sommerville, 2011, p. 84)

4. Modelos de Contexto

Los modelos de contexto son representaciones visuales de la interacción entre el sistema y su entorno, lo cual ayuda a comprender cómo se relacionará el software con otros sistemas o componentes externos. Estos modelos son fundamentales para identificar las entradas y salidas necesarias y los sistemas externos que el software debe soportar o con los que debe integrarse. Definir el contexto del sistema también permite establecer límites claros, reduciendo el riesgo de incluir funciones innecesarias o de perder funciones esenciales.

Los modelos de contexto suelen representarse mediante diagramas, como diagramas de flujo de datos o diagramas de interacción, que ilustran los procesos de comunicación entre el software y otros sistemas. La creación de estos modelos permite a los ingenieros analizar el entorno operativo del sistema y anticipar posibles problemas de integración. Al establecer el contexto, los ingenieros también pueden identificar cualquier restricción impuesta por otros sistemas o plataformas en los que el software debe ejecutarse. Estos modelos ayudan a prevenir problemas de interoperabilidad, asegurando que el software pueda funcionar correctamente en el entorno previsto. Además, un modelo de contexto detallado facilita la colaboración entre equipos de desarrollo, ya que todos tienen una representación visual común del sistema y sus relaciones. En el caso de software complejo, estos modelos también permiten organizar mejor el trabajo, definiendo los puntos de integración y los requisitos de compatibilidad.

Finalmente, los modelosde contexto contribuyen a mejorar la precisión en la fase de diseño y desarrollo, ya que establecen un marco de referencia claro para la construcción del sistema. (Sommerville, 2011, p. 121)

5. Patrones Arquitectónicos.

Los patrones arquitectónicos son soluciones probadas que organizan la estructura de un sistema de software, facilitando su desarrollo, escalabilidad y mantenibilidad. Estos patrones representan diseños estándar que los ingenieros pueden reutilizar en distintos proyectos. Uno de los patrones más conocidos es la arquitectura en capas, que organiza el software en niveles separados de funcionalidad, como la interfaz de usuario, la lógica de negocio y la capa de datos. Este patrón permite modificar una capa sin afectar las demás. Otro patrón ampliamente utilizado es el modelo cliente- servidor, que facilita la comunicación entre un cliente, que solicita información o servicios, y un servidor, que proporciona dichos servicios. Este patrón es ideal para sistemas distribuidos.

La arquitectura de microservicios es una variante que divide las aplicaciones en servicios independientes, facilitando la implementación, actualización y escalabilidad de cada componente sin afectar al resto del sistema.

En sistemas que requieren alta disponibilidad y escalabilidad, se emplea el patrón de arquitectura de eventos, que organiza el software en función de eventos que se activan en respuesta a acciones específicas, optimizando la respuesta en tiempo real. Los patrones arquitectónicos no solo ayudan en el diseño, sino que también mejoran la mantenibilidad y permiten a los ingenieros prever posibles problemas de integración entre módulos o sistemas.

Otro ejemplo de patrón es el modelo MVC (Modelo-Vista-Controlador), común en aplicaciones web, donde se separan los datos, la interfaz de usuario y la lógica de control, mejorando la organización y reutilización de componentes.

Estos patrones también facilitan la comunicación entre equipos, ya que los ingenieros comparten una base común de conocimientos que permite estructurar el software de manera predecible y coherente.

Los patrones arquitectónicos permiten a los desarrolladores implementar soluciones complejas basándose en estructuras probadas, reduciendo el tiempo de desarrollo y el riesgo de errores al reutilizar estrategias que ya han sido probadas y optimizadas. En resumen, los patrones arquitectónicos ofrecen una base sólida para desarrollar sistemas de software eficientes, escalables y fáciles de mantener, mejorando la calidad general del software y la eficiencia en el proceso de desarrollo. (Sommerville, 2011, p. 155)

6. Pruebas de Usuario

Las pruebas de usuario son un conjunto de actividades que evalúan la funcionalidad y usabilidad de un sistema desde la perspectiva del usuario final, ayudando a verificar que el software cumple con las expectativas y es fácil de usar. Uno de los principales objetivos es identificar problemas de usabilidad y funcionalidad antes de que el software sea lanzado al público, permitiendo a los desarrolladores realizar ajustes necesarios. Estas pruebas pueden incluir tareas específicas que los usuarios deben completar, evaluando cómo interactúan con la interfaz y si logran completar las acciones sin dificultades.

Las pruebas de usuario también son útiles para recoger comentarios directos sobre la experiencia del usuario, permitiendo a los desarrolladores comprender mejor las necesidades y expectativas de los usuarios.

Una ventaja de este tipo de pruebas es que permite identificar problemas que pueden haber pasado desapercibidos en las pruebas técnicas, especialmente aquellos relacionados con la interacción humana.

Los resultados de las pruebas de usuario son fundamentales para mejorar la usabilidad, ya que proporcionan datos concretos sobre los puntos de frustración o confusión dentro de la interfaz del software. Las pruebas de usuario pueden realizarse en diferentes etapas del desarrollo, desde prototipos iniciales hasta

versiones casi completas, permitiendo ajustes continuos durante todo el proceso.

Es importante que las pruebas de usuario se realicen con una muestra representativa del público objetivo para asegurar que los resultados reflejen las necesidades de los usuarios reales. Las pruebas de usuario contribuyen a la accesibilidad del software, ayudando a los desarrolladores a identificar y corregir barreras que puedan dificultar el uso por personas con discapacidades. En conclusión, las pruebas de usuario son una herramienta vital para asegurar que el software no solo cumple con los requerimientos técnicos, sino que también ofrece una experiencia positiva y accesible para todos los usuarios. (Sommerville, 2011, p. 228)

7. Calidad del Software

La calidad del software es una medida de la eficacia con la que el software cumple con los requisitos y expectativas del usuario, abarcando factores como funcionalidad, fiabilidad, usabilidad, eficiencia y mantenibilidad. Para garantizar un alto nivel de calidad, los ingenieros de software utilizan estándares y métricas que les permiten evaluar y mejorar cada aspecto del sistema de forma objetiva.

La funcionalidad evalúa si el software realiza todas las tareas requeridas correctamente, mientras que la fiabilidad mide la capacidad del software para funcionar de manera continua sin errores.

La usabilidad se refiere a la facilidad de uso y aprendizaje del software, asegurando que los usuarios puedan interactuar con el sistema sin dificultad y que les resulte intuitivo.

La eficiencia del software se mide en términos de velocidad y uso de recursos, optimizando el rendimiento del sistema para que funcione de forma rápida y no sobrecargue los dispositivos.

La mantenibilidad es esencial para el ciclo de vida del software, ya que asegura que el sistema sea fácil de actualizar y adaptar a nuevos requerimientos sin grandes esfuerzos. Además, la calidad incluye la seguridad del software, protegiendo los datos de los usuarios y previniendo accesos no autorizados que puedan comprometer la integridad del sistema.

Las pruebas y revisiones de código son esenciales para detectar y corregir problemas en la calidad del software, previniendo errores que puedan surgir en etapas posteriores del desarrollo.

Un sistema de alta calidad es más confiable y reduce el riesgo de fallos, mejorando la satisfacción del usuario y la longevidad del software en el mercado. En resumen, la calidad del software no solo se trata de cumplir requisitos funcionales, sino de garantizar que el sistema sea seguro, eficiente, fácil de usar y adaptable a futuros cambios. (Sommerville, 2011, p. 655)

8. Administración del Cambio

La administración del cambio es el proceso que gestiona de forma estructurada cualquier modificación en el software, asegurando que se realice sin comprometer su estabilidad o funcionalidad. Este proceso es crucial, ya que los requerimientos pueden cambiar con el tiempo debido a nuevas necesidades del cliente, cambios en el mercado o mejoras tecnológicas.

La administración del cambio incluye el control de versiones, permitiendo a los ingenieros rastrear y revertir modificaciones si es necesario, lo cual facilita la resolución de problemas y errores. Una buena administración del cambio permite realizar actualizaciones y mejoras en el sistema de forma gradual, sin afectar la experiencia del usuario ni comprometer la seguridad del software.

Este proceso también implica la documentación detallada de cada cambio, lo que facilita la comunicación entre los miembros del equipo y asegura que todos estén al tanto de las modificaciones implementadas.

La administración del cambio minimiza el riesgo de errores al implementar pruebas específicas para cada cambio, garantizando que no afecten otras partes del sistema. Además, permite planificar adecuadamente el despliegue de actualizaciones, evaluando el impacto de los cambios en el sistema y en la infraestructura de soporte.

La administración del cambio también involucra la gestión de la configuración, un proceso que mantiene un control centralizado de todos los elementos del sistema para facilitar la implementación de modificaciones.

La capacidad de gestionar cambios de forma eficiente contribuye a que el software pueda adaptarse a nuevos requisitos sin interrumpir el servicio, mejorando su sostenibilidad y adaptación al entorno. En conclusión, la administración del cambio es fundamental para mantener la continuidad y calidad del software, permitiendo a los desarrolladores adaptarse a los cambios de forma organizada y controlada. (Sommerville, 2011, p. 685)

Referencia:
Sommerville, I. (2011). Ingeniería de software (99 ed., L. M. Cruz Castillo, Ed.). Pearson Educación de México.

Guión Vídeo

Introducción.

(Durante todo el video habrá una melodía de piano. Primero aparece un saludo en pantalla seguido del título: "Introducción a la Ingeniería de Software: Principios y Prácticas Esenciales")

Narrador: Hola a todos, les doy la bienvenida a esta presentación. Como habrán visto, a lo largo de este tutorial se ha dado un desarrollo a la definición de la ingeniería de software, así que ahora, para dar más profundidad a este tema, en este video titulado "Introducción a la Ingeniería de Software: Principios y Prácticas Esenciales", exploraremos algunos de sus aspectos clave. Abordando temas que van desde la ética y métodos ágiles, hasta patrones arquitectónicos y la administración del cambio. Es importante mencionar que el contenido está basado en el texto de Ian Sommerville, titulado: "Ingeniería de Software", del año 2011. El cual sirve como una referencia completa para entender la complejidad y responsabilidad de esta disciplina.

Sin más preámbulo, comencemos.

1. Ética en la Ingeniería de Software

(En pantalla aparecerá una imágen relacionada a los subtemas y en la mención de cada uno estará el título correspondiente)

Narrador: Iniciemos hablando de la ética en la ingeniería de software, uno de los pilares que guía la práctica profesional en este campo. Los ingenieros de software tienen el deber de crear aplicaciones y sistemas que no solo cumplan con su propósito funcional, sino que también sean seguros y respetuosos con la privacidad de los usuarios.

Existen varias áreas donde la ética juega un papel crucial, esto son:

- **Seguridad y Confiabilidad:** Un ingeniero debe asegurarse de que el software es seguro y no presenta vulnerabilidades que puedan ser explotadas. En sectores como la salud o la aviación, cualquier error podría tener consecuencias graves.
- **Privacidad:** La protección de datos sensibles es esencial. Es responsabilidad del ingeniero proteger la información personal y evitar su uso indebido o exposición a ataques.
- **Transparencia y Honestidad:** Los ingenieros deben informar con claridad las limitaciones y riesgos asociados al software, evitando generar expectativas no realistas.
- **Sostenibilidad:** Minimizar el impacto ambiental es otro componente ético. Desde el diseño hasta el despliegue, los ingenieros deben buscar soluciones que reduzcan el consumo de recursos y energías.
- **Justicia y Respeto a la Propiedad Intelectual:** Respetar los derechos de autor y asegurar que el software no infringe la propiedad intelectual de terceros es fundamental.
- **Competencia Profesional:** La industria está en constante evolución. Por eso, los ingenieros deben actualizarse continuamente para responder a las demandas y retos actuales. (Sommerville, 2011, p. 14)

En resumen, la ética en ingeniería de software representa un compromiso importante para la sociedad, ya que asegura que el software creado beneficie a todos sin poner en riesgo su privacidad ni su seguridad.

2. Métodos Ágiles

(En pantalla aparecerá el título de cada subtema acompañado de una imágen, esto incluye una gráfica de Scrum y Extreme Programming XP)

Narrador: Ahora hablaremos de los métodos ágiles, los cuales revolucionaron la manera en que se desarrolla el software. Este conjunto de prácticas permite a los equipos adaptarse rápidamente a los cambios en los requisitos del cliente, promoviendo la flexibilidad y la colaboración continua. Entre los principios de los métodos ágiles, encontramos:

- **Entrega Incremental de Funcionalidades:** En lugar de desarrollar todo el sistema de una vez, los métodos ágiles dividen el proyecto en iteraciones, entregando pequeños componentes funcionales al cliente. Esto facilita la retroalimentación y permite realizar ajustes antes de la siguiente iteración.
- **Trabajo Colaborativo con el Cliente:** La interacción continua con el cliente garantiza que el desarrollo avance en la dirección correcta, ajustándose a las expectativas y necesidades que surgen.
- **Personas sobre Procesos y Herramientas:** La metodología ágil prioriza la comunicación directa y la colaboración entre los miembros del equipo.
- **Adaptación y Mejora Constante:** La revisión y ajuste de cada iteración permite responder a los cambios y mejorar continuamente el producto.

Entre los marcos de trabajo más populares están Scrum y Extreme Programming (XP), que brindan pautas estructuradas para organizar el desarrollo en equipos de trabajo. Scrum, por ejemplo, organiza el proyecto en “sprints”, o períodos de trabajo cortos, al final de los cuales se revisa y evalúa el progreso. Mientras tanto, XP hace énfasis en técnicas como la programación en parejas y la revisión de código frecuente, promoviendo la calidad y el aprendizaje continuo. (Sommerville, 2011, p. 58)

3. Requerimientos Funcionales y No Funcionales

(Aparecerá en pantalla los requerimientos “Funcionales” y “No Funcionales”, habrá imágenes relacionadas)

Narrador: Los requerimientos en la ingeniería de software son fundamentales, ya que describen las características y capacidades que debe tener el sistema. Los requerimientos se dividen en dos grandes categorías: funcionales y no funcionales.

- **Requerimientos Funcionales:** Definen las operaciones específicas que debe realizar el sistema. Por ejemplo, en un sistema bancario, uno de

estos requerimientos sería “permitir la transferencia de fondos entre cuentas”.

- **Requerimientos No Funcionales:** Se refieren a atributos de calidad como la velocidad de respuesta, la seguridad, y la facilidad de uso. Estos aspectos son igualmente importantes, ya que un sistema que cumple con todas sus funciones, pero es lento o difícil de usar, no será bien recibido.

El proceso de ingeniería de requerimientos incluye la captura y análisis de estos requisitos, la validación para asegurar que se comprendan correctamente, y la administración continua para adaptarlos a medida que el proyecto avanza. Este proceso ayuda a identificar posibles problemas en etapas tempranas, reduciendo los costos y mejorando la precisión en el desarrollo del sistema. (Sommerville, 2011, p. 84)

Cierre

(Aparecerá en pantalla las palabras “Gracias por ver”, acompañado de un gif animado)

Narrador: Con esto concluimos nuestro recorrido por los fundamentos de la ingeniería de software. Como hemos visto, desarrollar software es un proceso complejo que implica no solo habilidades técnicas, sino también un fuerte compromiso ético y organizacional. La ingeniería de software es clave para el desarrollo de aplicaciones y sistemas seguros, eficientes y fáciles de usar, que responden a las necesidades de la sociedad actual.

Esperamos que esta presentación te haya dado una comprensión clara de los aspectos esenciales de la ingeniería de software. Recuerda que este es un campo en constante evolución, y mantenerse actualizado es fundamental para seguir construyendo soluciones innovadoras y responsables.

Escaleta para Video: Fundamentos de la Ingeniería de Software

---

1. Introducción

- Visual: Animación ligera y música de fondo. Título en pantalla: "Introducción a la Ingeniería de Software: Principios y Prácticas Esenciales".
- Narración: Bienvenida e introducción al video. Explicación del objetivo y mención de la fuente principal (Ian Sommerville, *Ingeniería de Software*).

2. Ética en la Ingeniería de Software

- Visual: Imágenes relacionadas con cada subtema y aparición del título correspondiente.
- Narración: Explicación de los principios éticos clave en la ingeniería de software:
    - Seguridad y confiabilidad.
    - Privacidad.
    - Transparencia y honestidad. ○ Sostenibilidad. ○ Justicia y respeto a la propiedad intelectual.
    - Competencia profesional.

3. Métodos Ágiles

- Visual: Título de cada subtema acompañado de una imágen. Gráfica de Scrum y Extreme Programming XP.
- Narración: Introducción a los métodos ágiles y sus principios:
    - Entrega incremental. ○ Colaboración con el cliente. ○ Personas sobre procesos.

    - Adaptación y mejora continua.

- Ejemplos de metodologías (Scrum y Extreme Programming).

4. Requerimientos Funcionales y No Funcionales

- Visual: Título de ambos tipos de requerimientos con imágenes relacionadas.
- Narración: Diferenciación entre requerimientos funcionales (operaciones del sistema) y no funcionales (atributos de calidad). Descripción del proceso de ingeniería de requerimientos.

5. Cierre

- Visual: Despedida con las palabras “Gracias por ver” en pantalla, acompañado de un gif animado.
- Narración: Conclusión sobre la importancia de la ingeniería de software en la creación de sistemas seguros y eficientes. Llamado a seguir aprendiendo y mantener un compromiso con la ética y el profesionalismo.